Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Urban Baumkirher

Avtomatsko podnaslavljanje slik z globokimi nevronskimi mrežami

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Marko Robnik Šikonja

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Avtomatsko podnaslavljanje slik skuša v stavkih naravnega jezika opisati vsebino slike in dogajanje na njej. Tipično se za to nalogo uporabljajo konvolucijske nevronske mreže, ki jih naučimo na veliki zbirki že podnaslovljenih slik. Besedilne opise lahko uporabimo za semantično iskanje in združevanje slik z drugimi viri informacij. Preučite problem podnaslavljanja slik in implementirajte sistem, ki združuje konvolucijske nevronske mreže za zajem vizualnih značilk ter LSTM nevronske mreže za zajem besedilnih informacij iz že podanih opisov slik. Za učenje uporabite podatkovno zbirko podnaslovljenih slik MS CoCo. Sistem ustrezno ovrednotite in primerjajte z drugimi pristopi.

Za pomoč pri izdelavi diplomskega dela, koristne napotke, predlaganje literature, popravke, koordinacijo in usmerjevanje se iskreno zahvaljujem izr. prof. dr. Marku Robniku Šikonji.

Zahvaljujem se svoji družini za vse spodbudne besede in staršema za finančno podporo med študijem.

Še posebej se zahvaljujem dekletu Kaji, ki mi je z moralno podporo olajšala študijska leta in mi vedno znova dajala zagon za delo.

Svoji dragi Kaji.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CNN	convolutional neural network	konvolucijska nevronska mreža
RNN	recurrent neural network	rekurenčna nevronska mreža
LSTM	long-short term memory neural network	nevronska mreža z dolgoročnim in kratkoročnim pomnilnikom

Povzetek

Naslov: Avtomatsko podnaslavljanje slik z globokimi nevronskimi mrežami

Avtor: Urban Baumkirher

V diplomskem delu smo implementirali globoko nevronsko mrežo, ki smo jo naučili generirati stavčni opis slike. Mreža povezuje področje računalniškega vida in obdelave naravnega jezika. Sledili smo že objavljenim arhitekturam in arhitekturo implementirali s knjižnico Keras v jeziku Python. Podatke smo pridobili s spletne podatkovne zbirke MS COCO iz leta 2014. Naša rešitev implementira dvodelni model in uporablja globoke konvolucijske, rekurenčne in polno povezane nevronske mreže. Za obdelavo in zajem značilk slik smo uporabili arhitekturo VGG16. Besede smo predstavili z vektorsko vložitvijo GloVe. Model smo naučili na podatkovni zbirki 82.783 slik in testirali s 40.504 slikami ter opisi. Ocenili smo ga z mero BLEU in dosegli vrednost 49.0 ter klasifikacijsko točnost $\approx$ 60 %. Najboljših objavljenih rezultatov nismo dosegli, a obstaja še veliko možnosti za izboljšave.

Ključne besede: opisovanje slik, označevanje slik, strojno učenje, globoko učenje, nevronske mreže, konvolucijske nevronske mreže, rekurenčne nevronske mreže, LSTM mreže.

Abstract

Title: Automatic image captioning using deep neural networks

Author: Urban Baumkirher

We implemented a deep neural network, which we trained to generate image captions. The neural network connects computer vision and natural language processing. We followed existing architectures for the same problem and implemented our architecture with Keras library in Python. We retrieved data from an online data collection MS COCO. Our solution implements a bimodal architecture and uses deep convolutional, recurrent and fully connected neural networks. For processing and collecting image features we used the VGG16 architecture. We used GloVe embeddings for word representation. The final model was trained on a collection of 82.783 and tested on 40.504 images and their descriptions. We evaluated the model with the BLEU score metric and obtained a value of 49.0 and classification accuracy of ≈ 60 %. Current state-of-the-art models were not surpassed, but we see many possibilities for improvements.

Keywords: image captioning, machine learning, deep learning, neural networks, convolutional neural networks, recurrent neural networks, LSTM neural networks.

Poglavje 1

Uvod

Strojno učenje je področje umetne inteligence. Pomaga nam pridobivati znanje in vzorce v podatkih. Strojno učenje je uporabno na področjih medicine, biologije, kemije, podatkovnega rudarjenja, statistike, robotike, spletnih tehnologijah in ostalih raziskovalnih področjih [7].

Človeku preprost problem opisovanja slik računalniku predstavlja težko nalogo. Problem je sestavljen iz računalniškega vida in procesiranja naravnega jezika. V delih [6, 16] so za rešitev uporabili vizualne značilnosti slik in stavčne predloge. Z razvojem področja globokega učenja se danes problema lotevamo z nevronskimi mrežami. Uporabili smo dva različna tipa globokih nevronskih mrež, vsakega za svojo nalogo. Konvolucijske nevronske mreže uporabljamo za razpoznavanje vsebine slik in rekurenčne nevronske mreže za učenje tvorjenja opisov. Takšen pristop so uporabili v delih Andreja Karpathya [12] in v Googlovem projektu *Show and Tell* [28]. Deli smo vzeli kot zgled pri izdelavi podobne arhitekture.

Razvoj takšne tehnologije lahko pripomore na področju računalniškega vida pri prepoznavanju objektov, označevanju vsebine in klasificiranju slik, razvoju bogatejših priporočilnih sistemov, iskalnikov ipd.

Za učenje smo uporabili podatke iz zbirke *MS COCO*. S pomočjo uporabe že naučenega modela VGG16 smo izluščili značilke slik. Za učenje tvorjenja opisov smo uporabili rekurenčne nevronske mreže LSTM. Nevronsko mrežo,

ki je razdeljena na tri večje modele - slikovni, besedilni in združitveni model, smo implementirali v jeziku Python s knjižnico Keras. V modelu smo uporabili konvolucijske, rekurenčne in polno povezane nevronske mreže ter vektorske vložitve besed. Obdelane slike smo, skupaj s pripadajočimi opisi, uporabili za učenje modela. Preizkusili smo več kombinacij parametrov modela in tako izboljšali končni rezultat. Model smo ocenili z mero BLEU.

Vsebinski del diplomskega dela je razdeljen na 6 poglavij. V 2. poglavju spoznamo delovanje, teoretične osnove in različne vrste globokih nevronskih mrež. V 3. poglavju predstavimo programska orodja in knjižnice za izdelavo modela. V 4. poglavju opišemo arhitekturo modela, izdelanega v diplomskem delu. Spoznamo, kako deluje in na kakšen način smo prišli do rešitve. V 5. poglavju ovrednotimo model in predstavimo postopek iskanja optimizacije parametrov. Delovanje modela predstavimo na primerih slik. V 6. poglavju pregledamo narejeno, predstavimo ugotovitve in možne izboljšave.

Poglavje 2

Globoko učenje

V tem poglavju bomo spoznali področje globokega učenja, natančneje nevronske mreže in tipe nevronskih mrež, ki smo jih uporabili v končni arhitekturi modela za generiranje opisov slik.

2.1 Globoke nevronske mreže

Globoko učenje se je izkazalo za učinkovito na področjih prepoznave govora, obdelave zvoka, obdelave naravnega jezika, robotike, bioinformatike, kemije, video iger, v iskalnikih, spletnem oglaševanju in financah.

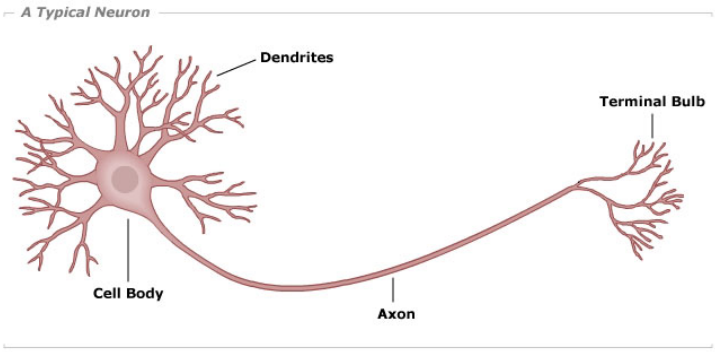
Izraz globoko učenje (ang. deep learning) se je pojavil leta 2000 kot *Deep Belief Nets*, kasneje, leta 2010, se je uveljavil danes poznani izraz [32]. Perceptron, ki je nastal že leta 1965, predstavlja osnovno gradbeno enoto nevronskih mrež. Globoke nevronske mreže zaradi takratnih strojnih omejitev niso v popolnosti zaživele. Globoke nevronske mreže, ki imajo večje število skritih slojev in vozlišč omogočajo, da iz preprostih konceptov zgradimo kompleksnejše [7]. S tehnološkim napredkom, so nevronske mreže zadnja leta doživele razcvet, saj imamo za učenje in napovedovanje dovolj zmogljive grafične procesorje (GPU-je), ki vsebujejo visoko stopnjo paralelizma in več tisoč procesorjev.

2.2 Nevronske mreže

Človeški možgani so eden najzmogljivejših bioloških učnih strojev in se uspešno prilagajajo notranjim in zunanjim dražljajem. Njihove dobre lastnosti so učenje, kratkoročen in dolgoročen spomin, možnost reševanja problemov s pomočjo intuicije, senzorično zaznavanje ipd.
Umetne nevronske mreže poskušajo posnemati človeške možgane. Možgani so sestavljeni iz mnogih živčnih celic, imenovanih nevroni, med katerimi se s pomočjo električnih impulzov prenašajo informacije. Na osnovi njihovega delovanja so zasnovane umetne nevronske mreže.

Perceptron

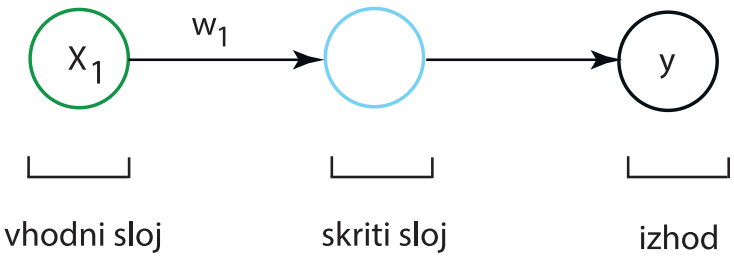
Perceptron je osnovna gradbena enota nevronskih mrež. Če vzamemo zgled človeških možganov, perceptron na Sliki 2.2 predstavlja nevron s Slike 2.1. Perceptron ima vhodne in izhodne povezave, kakor ima nevron vhodne dendride in izhodni akson z živčnimi končiči.



Slika 2.1: Risba tipičnega nevrona. Vir: [27].

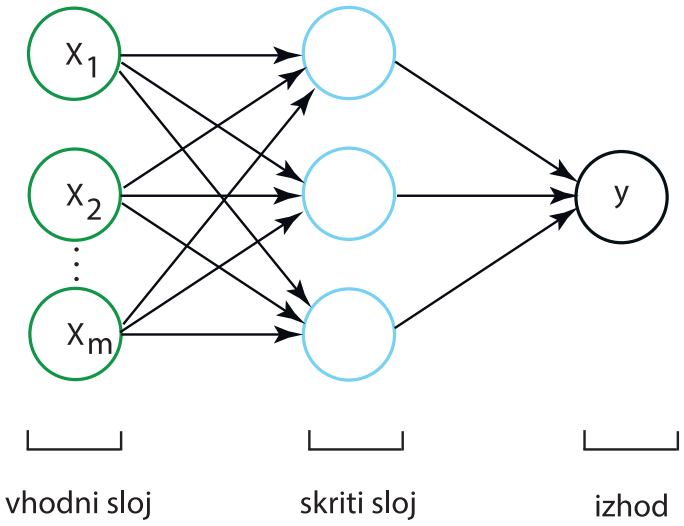
Perceptron sprejema informacije preko vhodov in se odloči, ali prejeta informacija izpolnjuje pogoje, da poda impulz naprej perceptronom, s katerimi je povezan njegov izhod. Vrednosti vhodov in izhodov perceptronov so števila, bioloških nevronov pa električni signal, ki ga v računalništvu mo-

deliramo kot vrednosti *da* ali *ne* (0 ali 1). Informacija na Sliki 2.2 potuje v smeri puščic – z leve proti desni oziroma iz vhodnega sloja proti izhodu. Posamezen atribut učnega primera naših podatkov predstavlja posamezno vozlišče na vhodnem sloju.



Slika 2.2: Diagram perceptrona z enim vhodom in izhodom.

En sam perceptron ni koristen, ko pa jih medsebojno povežemo več, ustvarimo zmogljivo mrežo povezanih perceptronov, imenovanih nevronska mreža (Slika 2.3). Takšna mreža bi navadno imela več vozlišč na vhodnem sloju in več vozlišč na skritem sloju. Globlja nevronska mreža bi imela še več skritih slojev.



Slika 2.3: Diagram preproste nevronske mreže.

Kadar imamo opravka z binarnim izhodom, zadošča, da imamo eno vozlišče na izhodnem sloju, kadar imamo n-razredno napoved, uporabimo n vozlišč na izhodnem sloju. Vsa vozlišča, razen vhodnih, hranijo za vsako vstopno povezavo (puščica) pripadajočo utež. Na Sliki 2.2 je označena z w_1. S pomočjo prilagajanja, uteži perceptron določa prioriteto oziroma pomembnost vhodnih signalov [21]. Perceptron se lahko prilagodi vhodnim vrednostim preko uteži. Pomembnost vhodov določa tako, da na določeni povezavi poveča utež ali izniči nekatere vhode z utežjo $w_i = 0$. Vrednost vhodnih signalov z utežmi za dani perceptron izračunamo z izrazom:

$$\sum_{i=1}^{m} x_i w_i, \tag{2.1}$$

kjer je m število vhodov, x_i vrednost i-tega vhoda in w_i vrednost uteži i-te povezave.

2.3 Aktivacijska funkcija

Ko smo izračunali vsoto produktov vhodnih vrednosti in uteži danega vozlišča (2.1), določimo vrednost izhoda. Za to poskrbi aktivacijska funkcija, ki določa zalogo vrednosti in vrednost izhoda vozlišča. Poznamo več različnih aktivacijskih funkcij, predstavili bomo tri najpogostejše. Aktivacijska funkcija se lahko spreminja skozi nevronsko mrežo, kar pomeni, da lahko vsakemu sloju posebej izberemo aktivacijsko funkcijo. Izbira je odvisna od problema. Najpreprostejša aktivacijska funkcija je *pragovna funkcija* (2.2).

$$\phi(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}, \tag{2.2}$$

kjer x predstavlja uteženo vsoto vhodov (2.1). Primerna je za uporabo na končnih izhodnih vozliščih nevronske mreže, kjer je pričakovana napoved binarna vrednost, saj se x preslika v vrednosti 0 ali 1 glede na podani prag (v tem primeru je prag 0).

Naslednja je *sigmoidna funkcija* (2.3) [18]:

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

Uporabna je pri preslikavi končnih napovedanih vrednosti v verjetnosti. V primerjavi s pragovno je bolj gladka, saj se od vrednosti 0 postopoma približuje vrednosti 1. Zaloga vrednosti sigmoidne funkcije so realna števila med 0 in 1.

Najpopularnejša aktivacijska funkcija je preprosta *pragovna funkcija* (2.4) (ang. rectifier function) ali linearna pragovna funkcija (ang. rectifier linear units). Vrednosti x, manjše od 0, se preslikajo v 0, ostale vrednosti ostanejo nespremenjene. Ta funkcija je pogosto uporabljena zaradi hitrosti izračuna in pri operacijah konvolucije na matričnih strukturah [9].

$$\phi(x) = max(0, x) \tag{2.4}$$

Perceptroni izračunajo svojo izhodno vrednost y_n kot:

$$y_n = \phi(\sum_{i=1}^{m} x_i w_i), \tag{2.5}$$

kjer vsoto produktov vhodnih vrednosti in uteži danega vozlišča vstavimo v izbrano aktivacijsko funkcijo in končno vrednost uporabimo kot izhodno informacijo [21].

2.4 Učenje

Učenje nevronske mreže poteka v več korakih. Podatki morajo imeti normalizirane vrednosti. Atributne n-terice (vrstice) podatkov predstavljajo vhodna vozlišča nevronske mreže. Kolikor atributov imamo, toliko vhodnih vozlišč mora imeti model nevronske mreže. Navadno podatke podajamo nevronski mreži na vhode v paketih določene velikosti. Izbira velikosti paketa vpliva na hitrost učenja in konvergence k iskani vrednosti (rešitvi). Manjši paketki (npr. 32) pomenjo, da bo učenje počasnejše, kot z večjimi paketki (npr. velikosti 512), vendar bo model verjetno hitreje skonvergiral. Nevronske mreže

za iskanje optimalnih uteži uporabljajo različne optimizacijske algoritme, ki s pomočjo gradientnega sestopa iščejo minimum cenilne funcije [18]. Najenostavnejša cenilka je definirana z izrazom:

$$C = \sum \frac{1}{2}(\hat{y} - y)^2, \tag{2.6}$$

kjer je $\hat{y}$ napovedana vrednost, y pa dejanska. Formula je vsota srednjih kvadratnih razlik med vrednostima, vrednost (ceno) funkcije želimo minimizirati.

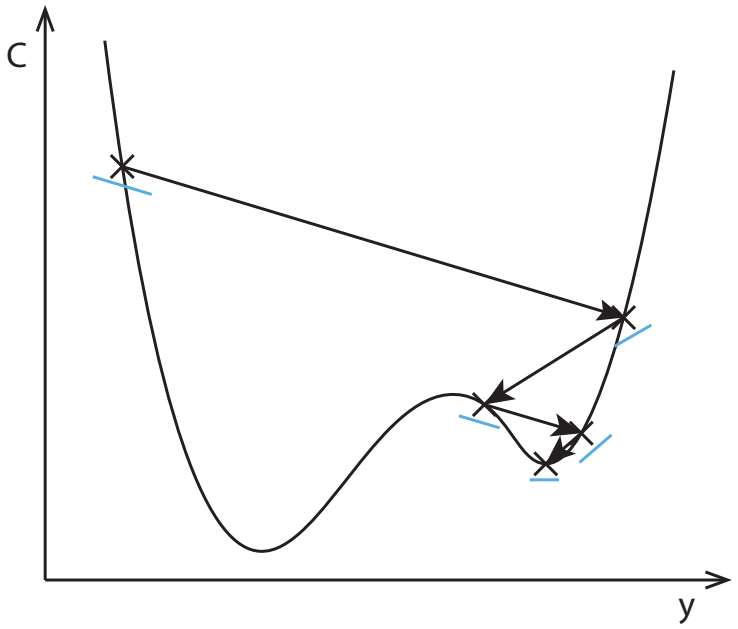
Prvi korak pri učenju nevronskih mrež je naključna inicializacija vseh uteži na vrednosti blizu 0, vendar ne na 0. Sledi dostava učnih podatkov v paketkih, s katerimi mreža po postopkih iz razdelka 2.3, izračuna končne izhodne vrednosti nevronske mreže (napovedi). Za vsako napovedano vrednost $\hat{y}$ v paketku mreža izračuna ceno s pomočjo cenilne funkcije (2.6). S postopkom vzvratnega učenja in gradientnega sestopa izračuna nove uteži in jih posodobi. Dodaten parameter, ki vpliva na velikost sprememb, se imenuje *učni faktor*. Koraki se ponavljajo, dokler nevronski mreži ne dostavimo vseh učnih podatkov. Končan cikel se imenuje *epoha* (ang. epoch). Učenje ponovimo za več epoh.

Gradientni sestop

Gradientni sestop je algoritem, ki se uporablja v procesu vzvratnega učenja. Cilj sestopa je konvergirati v točko, kjer cenilna funkcija dosega svojo najmanjšo vrednost (globalni minimum). Pristop k iskanju minimuma je preprost, zanj obstajajo številni optimizacijski algoritmi, kot so stohastični gradientni sestop, ADAM, RMSprop itd. Algoritmi stremijo k iskanju točke v prostoru, v kateri je strmina cenilne funkcije čim bližja ali enaka 0 [18, 33].

S Slike 2.4 je razvidno, kako se glede na naklon (modro obarvan vektor) algoritem premika po grafu funkcije. Iskanje največkrat zaznamuje cikcakasto pomikanje. Kjer je naklon velik, pomeni, da je vrednost cenilne funkcije precej oddaljena od minimuma.

Med iskanjem vrednosti se lahko pojavi več težav. Najbolj značilni sta:



Slika 2.4: Problem lokalnega minimuma pri iskanju minimuma cenilne funkcije z gradientnim sestopom.

divergenca - nakloni so preveliki in se v vsaki iteraciji pomaknemo preveč v levo ali desno in nikoli ne dosežemo boljšega rezultata. Rešitev tega problema je uvedba parametra *alfa*, ki določa faktor, kolikor naklona bomo upoštevali pri iskanju v naslednji iteraciji. V praksi parameter alfa zmanjšujemo skozi iteracije [18].

lokalni minimum - Slika 2.4 prikazuje ta problem. Rešujemo ga z naključjem. Iskanje razširimo in naključno preiskujemo več sestopov. Preizkušanje vseh možnih vrednosti ni mogoče. Problem rešujemo tako, da nevronski mreži dodamo več skritih slojev, lahko jim tudi povečamo število vozlišč. Ker vsako vozlišče na začetku preiskuje prostor naključnih vrednosti, smo rešili problem večkratnega naključnega iskanja v zameno za kompleksnejši model. S tem z večjo verjetnostjo najdemo globalni minimum problema.

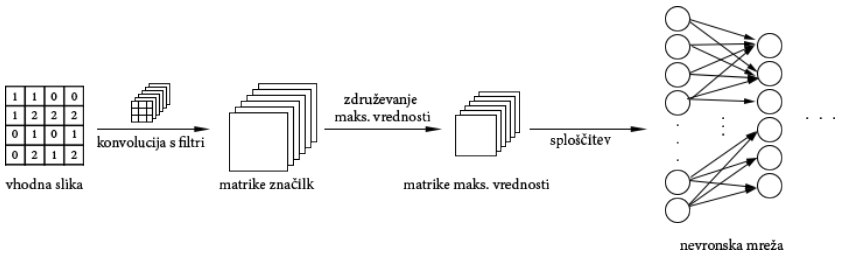
Izpust nevronov

Izpust nevronov (ang. dropout) je metoda, ki nevronskim mrežam z naključnim izničevanjem izhodnih vrednosti nevronov (pozabljanje) pomaga odpravljati problem prevelikega prileganja učnim podatkom (ang. overfitting). Navadno metodi izpusta nevronov določimo faktor ali odstotek, koliko naključnim nevronom bomo izničili (pozabili) vrednosti izhodov. Velik faktor izniči veliko število nevronov, vendar se lahko zaradi tega nevronska mreža premalo prilega učnim podatkom (ang. underfitting).

2.5 Konvolucijske nevronske mreže

Ustavimo se pri problemih prepoznavanja slik, vzorcev, zaporedij, signalov itd. Običajne nevronske mreže, kljub svoji prilagodljivosti, niso sposobne učinkovitega reševanja problema, kot je klasifikacija slik, saj je slika, predstavljena s točkami, preveč kompleksna. Zato potrebujemo drugačen zapis slik, da bi se nevronska mreža naučila klasificiranja njihove vsebine. V ta namen so bile razvite konvolucijske nevronske mreže (CNN), prikazane na Sliki 2.5.

CNN so pravzaprav nadgrajene polnopovezane nevronske mreže. Razlika



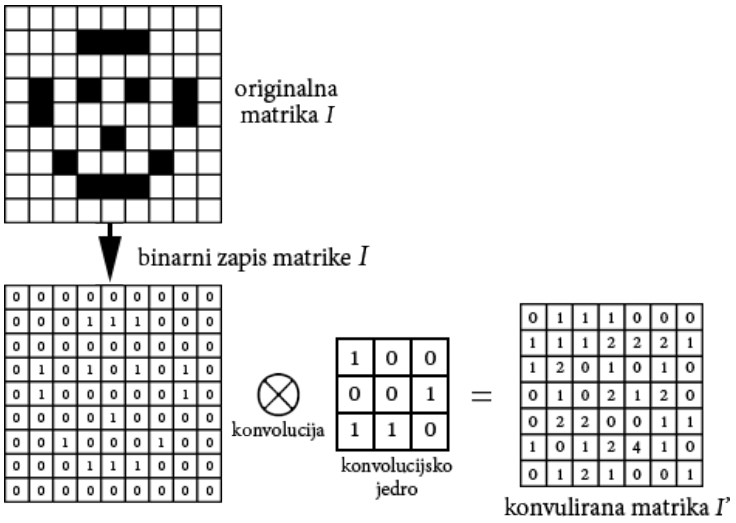
Slika 2.5: Splošna arhitektura konvolucijske nevronske mreže.

je v predprocesiranju vsebine vhodov. CNN temeljijo na matematičnem pojmu konvolucije, združevanju maksimalnih vrednosti in sploščitvi, s čimer dosežemo kodiranje vsebine za vhod v običajno nevronsko mrežo [17]. V tem

poglavju bomo spoznali naštete pojme in delovanje CNN.

Konvolucija

Matematična konvolucija odraža medsebojen vpliv dveh funkcij ali preprosteje – na kakšen način ena funkcija modificira drugo.
Za različne tipe podatkov lahko v CNN uporabimo različne konvolucije: enodimenzionalne za kodiranje zaporedij besed ali raznih signalov, dvodimenzionalne za slike in trodimenzionalne konvolucije, ki se uporabljajo za video - tretjo dimenzijo predstavlja čas. Za izdelavo rešitve v diplomski nalogi smo



Slika 2.6: Posplošen postopek konvolucije binarne slike.

uporabili 2D konvolucijo za analizo slik, kot je prikazano na sliki 2.6. Konvolucijsko jedro korakoma pomikamo po matriki I od levega roba matrike proti desnemu. Korak (ang. stride) je poljuben, ponavadi je enak dolžini konvolucijskega jedra. Na vsakem koraku izračunamo konvulirane vrednosti z izrazom (2.7) in jih zapišemo v novo konvulirano matriko I'. Ko dosežemo desni rob matrike I, pomaknemo konvolucijsko jedro skrajno levo in navzdol za višino konvolucijskega jedra [33]. Ta postopek lahko matematično

zapišemo kot preprosto linearno konvolucijo:

$$I'(u,v) = \sum_{(i,j)\in R_H} I(u+i, v+j)H^*(i,j), \tag{2.7}$$

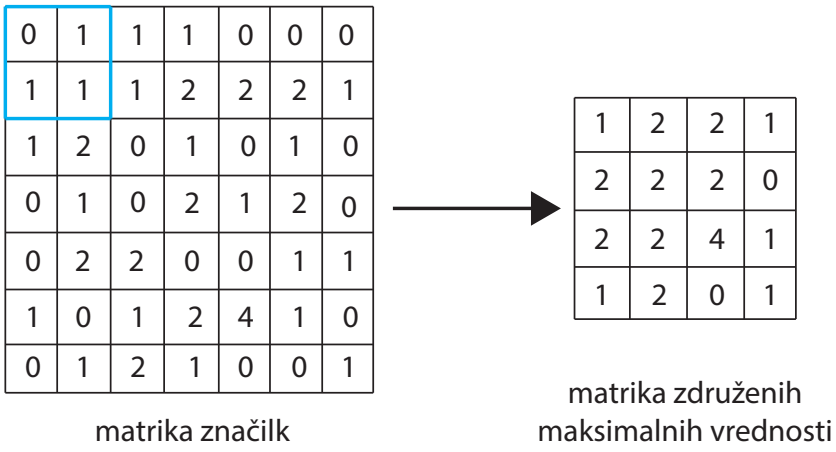
kjer je I originalna matrika slike dimenzij $u * v$, H^* zrcaljena matrika konvolucijskega jedra (filtra) dimenzij $i * j$, R_H področje konvolucijskega jedra in I' konvulirana matrika slike.
Slike so tipično zapisane v trodimenzionalni matriki, kjer sta dve dimenziji širina in višina slike, tretja dimenzija so trije barvni kanali (RGB, oziroma rdeča, zelena in modra). CNN uporabljajo različna konvolucijska jedra, s katerimi transformiramo originalno vhodno matriko v manjšo, ki hrani informacijo o značilkah slike. Rešitev si lahko predstavljamo intuitivno – človeški možgani na sliki najprej zaznajo specifične značilnosti, s katerimi nadalje razpoznamo objekt na sliki. Z večjim številom konvolucijskih jeder pridobimo zbirko matrik, ki vsebujejo značilke slike.

Združevanje maksimalnih vrednosti

Združevanje z maksimalno vrednostjo (ang. max pooling) je postopek, s katerim izluščimo najočitnejše značilke iz konvuliranih matrik značilk. Izberemo velikost drsečega okvirja filtra, znotraj katerega bomo iskali maksimalne vrednosti značilk. Postopek prikazuje Slika 2.7.

Tako zagotovimo zmanjšanje matrik, kar pripomore k hitrejšemu procesiranju, pridobimo tudi prostorsko invarianco. CNN tako izboljša učenje, saj bo slika s kakršnokoli linearno transformacijo prepoznana enako kot original [33]. Postopek združevanja z maksimalno vrednostjo ni brez izgub, vendar želimo izgubiti nepomembne značilke in ohraniti najizrazitejše.

Drseče okno dimenzij $2 * 2$ (na Sliki 2.7) na enak način, kot konvolucijsko jedro, pomikamo po matriki značilk. Izbrali smo pomikanje s korakom 2. V vsakem pomiku (koraku) poiščemo maksimalno vrednost znotraj drsečega okna in ga zapišemo v matriko združenih maksimalnih vrednosti.



Slika 2.7: Primer združevanja z maksimalom matrike značilk s Slike 2.6 z oknom (obarvan modro) dimenzij $2 * 2$ in korakom 2.

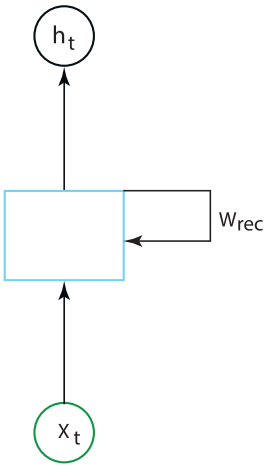
Sploščitev in polna povezava

Sploščitev je zadnji korak predprocesiranja vhodnih podatkov, preden dodamo polno povezane nivoje. S konvolucijo in združevanjem z maksimalno vrednostjo smo izluščili značilke slik in s tem zmanjšali količino podatkov za obdelavo. Končno zbirko matrik z značilkami s postopkom sploščitve preoblikujemo v enodimenzionalni vektor. Vektor posredujemo na vhod običajni nevronski mreži [33].

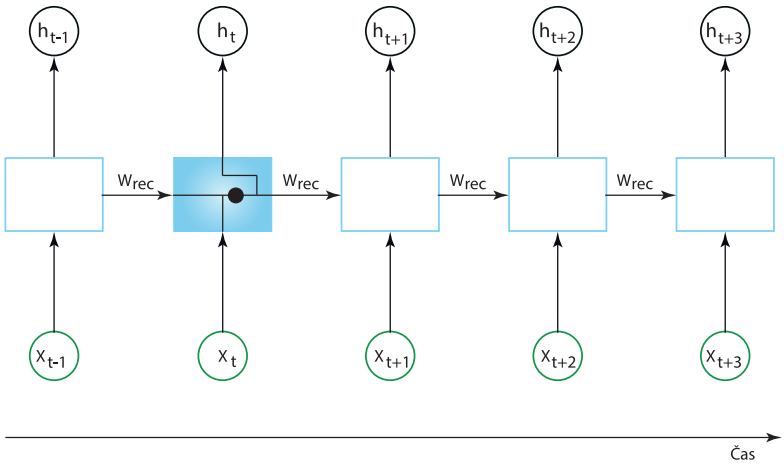
2.6 Rekurenčne nevronske mreže

Poglejmo si še postopek razumevanja in obdelave naravnega jezika. Ljudje beremo besedo za besedo, pri tem potrebujemo kratkoročen spomin, da razumemo kontekst trenutne besede v stavku. Zavedati se moramo, kaj smo ravnokar prebrali, da razumemo prebrano. Gre za problem obdelave in vsebinskega razumevanja zaporedij, pri čemer potrebujemo trajnost informacije. V ta namen uporabimo rekurenčne nevronske mreže (ang. recurrent neural networks, RNN). Če običajne nevronske mreže predstavljajo osnovno delovanje

možganov, konvolucijske mreže pa človeški vid v zatilnem režnju možganov, potem rekurenčne nevronske mreže modelirajo čelni reženj možganov, kjer se nahaja naš kratkoročni spomin. RNN so zato najprimernejše za obdelavo zaporedij, ki se pojavijo v problemskih domenah, kot so prepoznavanje govora, modeliranje jezika, prevajanje in problem opisovanja slik [23].



Slika 2.8: Diagram ene celice v rekurenčni nevronski mreži.



Slika 2.9: Diagram rekurenčne nevronske mreže.

Slika 2.8 predstavlja osnovno obliko RNN. Sposobnost za ohranjanje in-

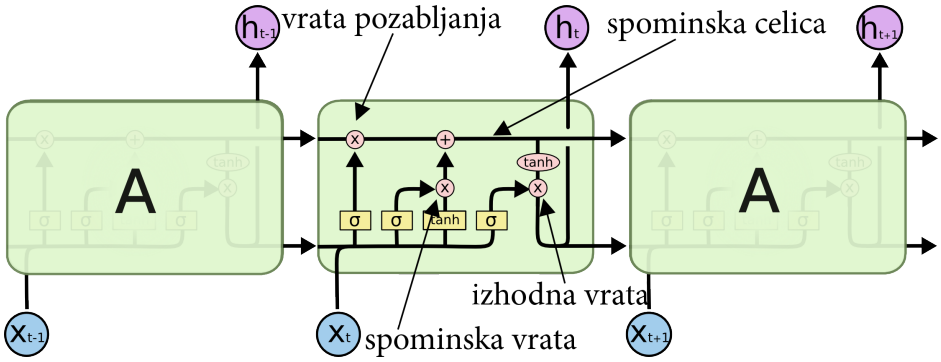
formacije med učenjem zaporedij omogoča ciklična povezava, ki je na Sliki 2.8 označena z utežjo W_{rec}. Poleg izhoda h_t se informacija obnovi s časovnim zamikom preko povezave W_{rec}. Podajanje informacije je prikazano na Sliki 2.9, kjer smo RNN razprli skozi čas. Z vztrajanjem podatka na vhodu, dosežemo podoben učinek, kot ga ima kratkoročni spomin, saj lahko RNN iz že videnega zaporedja napove naslednji najverjetnejši element, ki sledi zaporedju. RNN so dobre pri napovedovanju krajših zaporedij. Ker skozi čas ohranjamo informacijo le o nekaj prejšnjih elementih zaporedja, naletimo na problem dolgotrajnih odvisnosti [23]. Če RNN uporabimo za problem napovedovanja naslednje besede v kratkem stavku, bo to naredila brez težav. Kadar pa se soočimo z daljšim zaporedjem, kot je celo besedilo, takšna arhitektura ne omogoča dovolj dolgega pogleda nazaj, da bi poiskali dolgoročne odvisnosti med besedami iz začetka in besedami s konca. Problem rešujemo s posebno obliko RNN, ki jo označujemo z LSTM.

Problem pojemajočega in rastočega gradienta

Problem se pojavi med učenjem rekurenčnih nevronskih mrež. Denimo, da opazujemo vozlišče na 4. koraku Slike 2.9. Ko se izvaja postopek vzvratnega učenja, se bodo posodobitve uteži izračunale za vse prejšne korake. Ker je utež W_{rec} **ista** za vsako vozlišče, se lahko zgodi, da ob izračunu gradienta pri vsakem koraku nazaj dobimo vse manjše vrednosti. Pojav pojemajočega gradienta nastopi, kadar je vrednost $W_{rec} < 1$. Kadar je $W_{rec} > 1$, govorimo o problemu rastočega gradienta, saj sčasoma dobimo vse večje vrednosti gradienta [8]. Oba problema sta kritična za učenje RNN. Intuitivno to pomeni, da majhen (skoraj ničelen) gradient vozlišča v koraku 1 povzroči, da se to vozlišče ne bi učilo, ali pa bi se učilo prepočasi. Težava se posledično propagira naprej, saj je korak 4 odvisen tudi od koraka 1. Problem rešujemo tako, da nastavimo $W_{rec} \simeq 1$, ali uporabimo rekurenčne nevronske mreže LSTM.

Nevronske mreže LSTM

LSTM (ang. long short-term memory) je posebna oblika RNN, ki rešuje problem izginjajočega gradienta in težavo dolgoročnih odvisnosti, saj je sposobna ohraniti informacijo dalj časa. LSTM s pomočjo modificiranja vhodov uspe določiti, katere podatke podati naprej, katere upoštevati in jih spremeniti. Mreža uporablja dodatna vozlišča, ki spreminjajo in delijo vhod. Ima dodaten vhod, imenovan spominska celica, ki informacijo prenaša skozi čas. Spominski celici lahko glede na trenutno stanje vozlišča pripne informacijo s spominskimi vrati ali jo odstrani z vrati pozabljanja. Z izhodnimi vrati določa količino in del informacije, ki se bo iz spominske celice prenesla na izhod v naslednje vozlišče rekurenčne nevronske mreže [8, 13, 23].



Slika 2.10: Diagram rekurenčne nevronske mreže LSTM. Vir: [23].

Rumeni pravokotniki znotraj spominske celice rekurenčne nevronske mreže LSTM na Sliki 2.10, predstavljajo operacije nad sloji (npr. sigmoidna aktivacijska funkcija). Rdeči krogci predstavljajo računske operacije vektorjev kot sta seštevanje in množenje [23].

Poglavje 3

Orodja

V tem poglavju predstavimo programsko opremo, ki smo jo uporabili za razvoj modela za generiranje opisa iz slik. V programskem jeziku *python* smo s pomočjo knjižnic pickle, pandas in numpy, pripravili podatke. S knjižnico *Keras* smo implementirali končni model. S programsko opremo CUDA toolkit smo poganjali program učenja modela, neposredno na grafični procesni enoti.
Anaconda je priljubljena distribucija programskega jezika python s paketi, ki so namenjeni obdelavi podatkov. Vsebuje tudi orodja za analizo velepodatkov. Poleg namestitve distribucije pythona z orodji in knjižnicami omogoča paketno urejanje nameščenih programov, ki so ponujeni v paketih. Paket vključuje najnovejšo distribucijo pythona, urejevalnik Spyder, Jupyter in nekatere pythonove knjižnice [1]. Anacondo smo uporabili za namestitev potrebne programske opreme.

3.1 Python in Keras

V diplomskem delu smo uporabili programski jezik Python, ki je zaradi velikega števila zunanjih knjižnic primeren za strojno učenje. Skupaj s knjižnicami, kot so *Numpy*, *pandas*, *pickle* ter *Keras* nam je jezik omogočil hitro prototipiranje, uporabo ter pripravo podatkov in vzpostavitev končne

arhitekture modela. Na kratko spoznajmo naštete knjižnice:

NumPy - je knjižnica, namenjena znanstvenim izračunom in podpira zmogljive N-dimenzionalne tabele za linearno algebro. Uporaba objektov NumPy pospeši računske operacije v pythonu [2]. S knjižnico enostavno definiramo matrike, s katerimi pogostokrat operiramo v diplomskem delu.

pandas - je knjižnica, namenjena podatkovni analizi. Ponuja hitre in fleksibilne podatkovne strukture, namenjene delu z relacijskimi in označenimi podatki na enostaven način [3].

pickle - je knjižnica, namenjena binarni serializaciji in deserializaciji objektov v pythonu. Omogoča hitro shranjevanje in branje objektov iz datoteke in pretvorbo v pythonove objekte [4]. V diplomskem delu smo knjižnico uporabili za shranjevanje večjih objektov, ki jih ni bilo smiselno kreirati ob vsakem zagonu.

CUDA toolkit - skupek programskih orodij, vključno z gonilniki CUDA, ki omogočajo uporabnikom *NVIDIA* grafičnih kartic uporabo in pospešitev paralelnih izračunov za učenje nevronskih mrež z jedri grafične kartice [22].

Keras - je knjižnica za hitro prototipiranje nevronskih mrež. Podpira konvolucijske in rekurenčne nevronske mreže, ki smo jih uporabili. Keras ovija tri knjižnice za strojno učenje: *Tensorflow*, *Theano* in *CNTK*. S pomočjo Kerasa izberemo, katero izmed knjižnic bomo uporabili za delovanje zalednega sistema. Omogoča modularno uporabo podprtih knjižnic z manj vrstic programske kode in klicev funkcij [15]. Keras ob namestitvi CUDA gonilnikov omogoča uporabo grafične kartice za učenje modelov, kar pospeši proces.

Poglavje 4

Podatki in arhitekture

V tem poglavju se bomo seznanili z učnimi podatki. Predstavili bomo arhitekturo, ki smo jo implementirali v knjižnici Keras in razložili delovanje posameznih komponent arhitekture.

4.1 Podatki

Za učenje modela smo uporabili MS COCO (Microsoft Common Objects in Context) zbirko podatkov, ki vsebuje 328 tisoč slik [19, 20]. Od teh smo uporabili 128.287 slik z opisi. Zbirka je namenjena modernim metodam detekcije objektov. Sestavljajo jo preproste in kompleksne slike iz vsakodnevnih kontekstov. Zbirka ponuja veliko količino podatkov in raznolike vsebine. Slike, ki sestavljajo zbirko, so bile zbrane s spletne galerije *Flickr*. Za vsako sliko ima zbirka klasificirane objekte, semantično segmentacijo, lokalizacijo objektov, anotacije in stavčne opise [19]. Z uporabo vmesnika *COCO API* lahko nalagamo in urejamo dodatne informacije o slikah. Informacije smo brez uporabe vmesnika pridobili v *json* formatu s spletne strani, namenjene opisovanju slik, ki jo vodi Andrej Karpathy [14]. Primer dodatnih informacij o naključni sliki iz zbirke:


```
[{"filepath": "val2014", "filename": "COCO_val2014_000000391895
.jpg", "sentences": [{"tokens": ["a", "man", "with", "a", "red"
```

, "helmet" ..

Zbirka slik je dostopna na spletni strani MS COCO [20]. Razdeljena je na učno (82783 slik) in testno/validacijsko (40504 slik) množico. V diplomskem

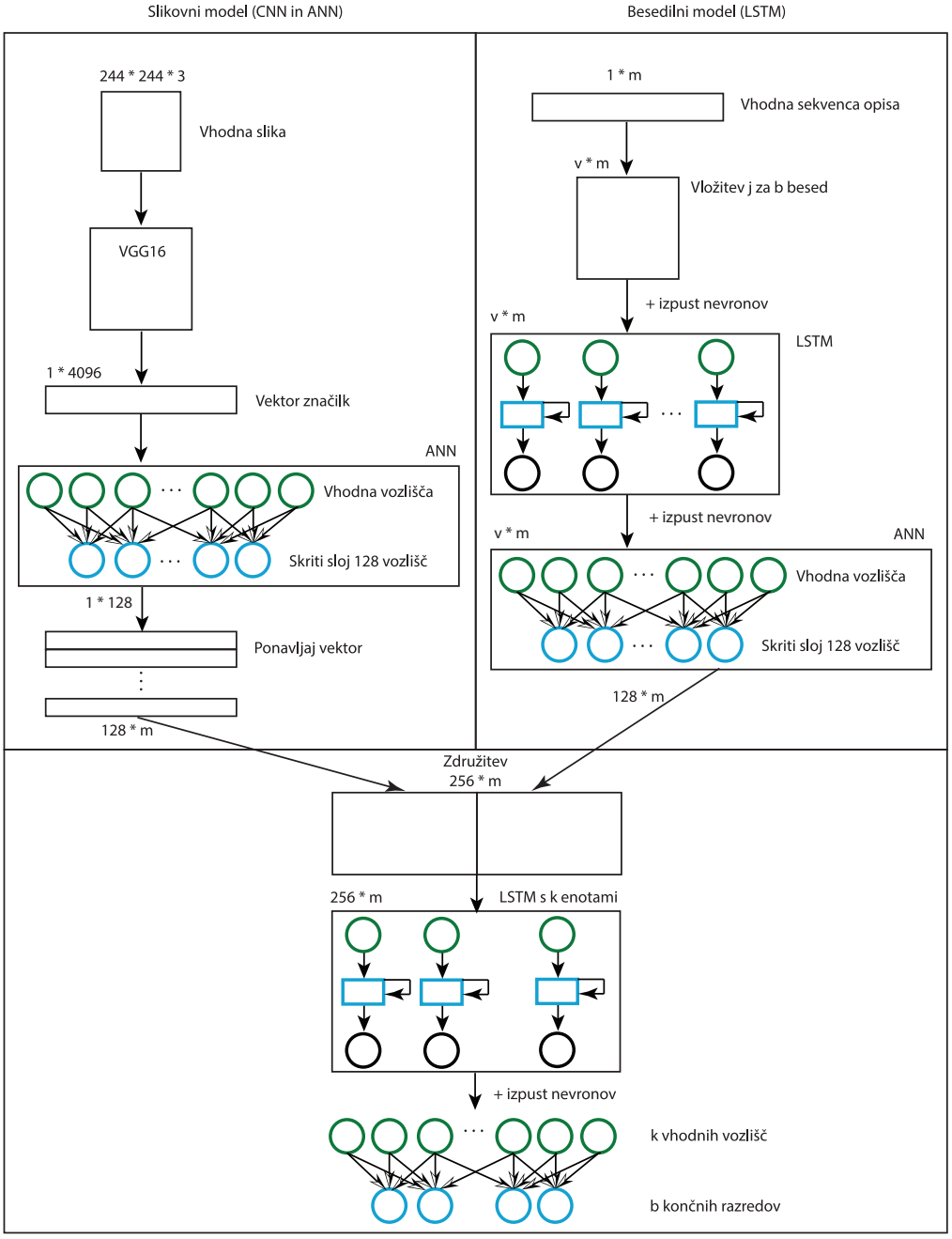
Slika 4.1: Primeri naključnih slik iz zbirke MS COCO.

delu smo iz zbirke podatkov uporabili le slike in pripadajoče opise v angleškem jeziku. Za posamezno sliko je 5 opisov. Zaradi časovne zahtevnosti učenja modela smo se odločili, da uporabimo samo prvi opis vsake slike.

4.2 Arhitektura nevronske mreže

Problem avtomatskega opisovanja slik povezuje področje računalniškega vida in procesiranja naravnega jezika. Potrebovali smo model, ki na vhod prejme sliko in pripadajoč opis. Uporabili smo dve vrsti nevronskih mrež: konvolucijske nevronske mreže za razbiranje vsebine (značilke) iz slik in rekurenčne nevronske mreže za napovedovanje zaporedij. Vhod smo razdelili na dva sklopa, vsak sklop prenaša vhodne podatke v svoj tip nevronske mreže. Podobno arhitekturo so za isto problemsko domeno uporabili v delu Andreja Karpathya [12] in v Googlovem raziskovalnem projektu *Show and Tell* [28]. Na sliki 4.2 smo model ločili na slikovni in besedilni model. Na Sliki so nad vsako manjšo komponento (kvadrat ali pravokotnik) zapisane dimenzije vhodnih podatkov. Če se dimenzije vhodnih podatkov na izhodu spremenijo, je nova dimenzija izhoda napisana ob izhodni puščici.
V slikovnem modelu smo uporabili *VGG16* arhitekturo konvolucijske nevronske mreže, ki je podrobneje razložena v razdelku 4.2.1. Konvolucijska

nevronska mreža VGG16 je namenjena klasificiranju slik, mi smo model preoblikovali tako, da smo njegov zadnji sloj, ki napoveduje verjetnosti razredov, odstranili ter izhod nastavili na predzadnji sloj. Predzadnji sloj nam je tako namesto verjetnosti razredov, vračal vektor dolžine 4096, na Sliki 4.2 označen z *vektor značilk*, ki predstavlja vrednosti značilk vhodne slike. Enak pristop izluščitve značilk slike je bil uporabljen v delu Karpathya [12]. Značilke so vhod v polnopovezano nevronsko mrežo, ki je vektor značilk preoblikovala v vektor dolžine 128. Tega, zaradi potrebe po ujemanju dimenzij izhodov pri združitvi obeh modelov, ponavljamo. Na ta način smo zmanjšali količino podatkov in poglobili model za boljše prileganje značilkam slike.
Besedilni model sestavljajo sloji vložitev besed, rekurenčna nevronska mreža LSTM in polno povezana nevronska mreža (razloženi v razdelku 4.2.2). Na vhod je besedilni model prejel sekvence opisa, kar pomeni, da smo ga postopoma – besedo za besedo, naučili tvoriti stavek. Tvorjenje sekvenc je razloženo v razdelku 5.1. Vhodno sekvenco vložitvenih vektorjev smo preoblikovali v vektorje, ki v večdimenzionalnem prostoru nevronski mreži predstavljajo odvisnosti med besedami. Na vložitvi smo uporabili naključno izpuščanje nevronov (ang. dropout). Vložitev trenutne sekvence je vhod v rekurenčno nevronsko mrežo LSTM. Ponovno smo uporabili izpuščanje nevronov ter vsak izhod neodvisno povezali s polnopovezano nevronsko mrežo. Izhoda besedilnega in slikovnega modela smo združili (konkatenacija matrik) in rezultat zadnjič obdelali z rekurenčno nevronsko mrežo LSTM, njene izhode pa pripeli na vhod polno povezane nevronske mreže, ki ima toliko izhodnih vozlišč, kolikor je velikost slovarja besed. Na kratko: na prvi vhod smo postavili sliko, na drugega pa sekvence opisa. Kot izhod nam je model vrnil verjetnosti za naslednjo besedo, ki naj bi sledila trenutni vhodni kombinaciji sekvence opisa in slike.



b- velikost slovarja besed (število razredov)
k- število LSTM enot
m- dolžina najdaljšega opisa
v- dolžina vložitvenega vektorja

Slika 4.2: Podrobna arhitektura modela.

4.2.1 Slikovni model

V tem razdelku bomo spoznali podrobnosti slikovnega modela s Slike 4.2. Naloga slikovnega modela je pretvoriti vhodno sliko v matriko vrednosti značilk, ki predstavljajo vsebino (objekte) slike. Takšno kodiranje slik smo dosegli z več konvolucijskimi sloji in združevanjem z maksimumom z uporabo VGG16 modela.

Arhitektura VGG16 modela je prikazana na Sliki 4.3. Vhod slikovnega modela je slika, predstavljena v matriki dimenzij $244 * 244 * 3$, kar je zahteva VGG16 arhitekture. Vektorje značilk vhodne slike, pridobljenih iz VGG16 modela, smo poslali v polno povezano nevronsko mrežo, ki je pripomogla k temu, da se vozlišča nevronske mreže prilagodijo značilkam in zmanjšajo dolžino vektorja na 128 značilk. Ker je izhod *besedilnega modela* matrika, smo morali izhodni vektor slikovnega modela na koncu s ponavljanjem pretvoriti v matriko, da smo zagotovili pravilne dimenzije pri združevanju izhoda obeh modelov:

```
from keras.layers import Dense, RepeatVector, Input
#Slikovni model
#Vhodno vozlišče, za vektor značilk VGG16 modela
image_input = Input(shape=(4096,), name='img_input')
image_conn = Dense(units = 128, activation='relu',
name='img_fc2')(image_input)
repeat_img = RepeatVector(max_caption_len,
name='img_out_repeat')(image_conn)
```

Konvolucijska nevronska mreža VGG16

VGG16 (Visual Geometry Group) je globoka 16-slojna arhitektura konvolucijske nevronske mreže, ki jo je ekipa *Visual Geometry Group* z univerze v Oxfordu izdelala za nalogo lokalizacije in klasifikacije slik v *ImageNet* izzivu leta 2014 [24, 25]. 16-slojni globoki konvolucijski nevronski mreži VGG16 so zmanjšali število uteži tako, da so v vsakem konvolucijskem sloju uporabili

manjša konvolucijska jedra dimenzij 3*3 s korakom 1 (ang. stride).
ImageNet je organizirana podatkovna zbirka 1.419.7122 slik, ki so bile ročno kategorizirane [10]. VGG16 arhitektura je na takratnem izzivu dosegla prvo in drugo mesto. Arhitektura dobro deluje tudi na drugih podatkovnih zbirkah, zato smo se odločili, da jo uporabimo kot del slikovnega modela. Arhitektura VGG16 je postala standardna konvolucijska nevronska mreža knjižice Keras, kjer je implementirana. Skupaj z inicializacijo arhitekture VGG16 v Kerasu smo v model naložili že naučene uteži, ki so bile uporabljene na ImageNet izzivu.

Slika 4.3: VGG16 sloji arhitekture. Vir: [26].

VGG16 arhitekturi smo odstranili zadnji sloj in prevezali izhod na predzadnjega, da smo pridobili vektor značilk slike. Izsek pythonove kode nalaganja

modela VGG16 in prevezave izhodov:

```
import vgg16 as vg

def load_model():
    model = vg.VGG16(include_top=True, weights='imagenet')
    #modelu odstranimo softmax sloj (odstranimo napoved razreda)
    model.layers.pop()
    #modelov output prevezemo na predzadnji sloj (od zdaj zadnji)
    model.outputs = [model.layers[-1].output]
    model.layers[-1].outbound_nodes = []
    return model
```

4.2.2 Besedilni model

Besedilni model je zadolžen za kodiranje zaporedja besed v numerične vrednosti. Potrebovali smo metodo vložitve besed (ang. embedding), ki dano besedo preslika v vektorski prostor. Vložitev deluje kot poizvedovalna tabela, v kateri hranimo n-dimenzionalne vložitvene vektorje. Vektor je lahko poljubne dolžine, vendar smo za natančnejšo predstavitev besede uporabili vektorje večjih dolžin. V diplomskem delu smo uporabili GloVe (Global Vectors for Word Representation), ki so vektorji, pridobljeni z nenadzorovanim strojnim učenjem na različnih korpusih [11]. GloVe vektorji so različnih dolžin, mi smo uporabili ponujene vektorje dolžine 200, naučene na korpusu *Wikipedije* iz leta 2014. Značilnost vložitvenih vektorjev je, da so si semantično in tematsko podobne besede v vektorskem prostoru blizu. Na ta način smo modelu omogočili, da je združil podobne besede. GloVe vektorji poskušajo zajeti dovolj natančno informacijo o preslikanih besedah, zato semantično podobne besede ležijo blizu skupaj (v skupinah ali gručah) v vektorskem prostoru.

GloVe vektorje smo v arhitekturi uporabili kot predhodno naučene uteži vložitvenega sloja v besedilnem modelu na Sliki 4.2. GloVe vložitvene vektorje smo modelu podali kot uteži:

```
from keras.layers import Embedding, Input

#Vhodno vozlišče
caption_input = Input(shape=(max_caption_len, ),
name='cap_input')
#Vložitev z GloVe utežmi
embedd = Embedding(vocab_size, embedding_vec_len,
input_length = max_caption_len, mask_zero = True,
weights=[trained_embeddings],trainable=True,
name='embedding')(caption_input)
```

S knjižnico Keras smo inicializirali vhodno vozlišče in mu določili obliko vektorja vhoda. Nato smo vhodnemu vozlišču pripeli še vložitveni sloj (Embedding) in s parametrom *weights* naložili matriko GloVe vektorjev besed. Dodali smo še rekurenčno nevronsko mrežo LSTM, ki se je iz danega zaporedja vloženih besed učila tvorjenja stavkov in kako si besede sledijo v stavkih angleškega jezika:

```
#Manjši dropout vložitve
embedd_dropout = Dropout(0.1)(embedd)
#LSTM
cap_lstm = LSTM(embedding_vec_len,
return_sequences=True,
name='many2many_lstm')(embedd_dropout)
#LSTM dropout
lstm_dropout = Dropout(0.1)(cap_lstm)
#ANN vsak korak v času
cap_dense = TimeDistributed(Dense(128),
name='independent_dense')(lstm_dropout)
```

Na koncu sloja LSTM smo vozliščem v vsakem časovnem koraku s funkcijo *TimeDistributed* pripeli polno povezano nevronsko mrežo in s tem poglobili model, posledično povečali prilagodljivost modela in kvaliteto učenja zaporedij ter zmanjšali dolžino izhodnega vektorja.

4.2.3 Združitev modelov

Izhoda slikovnega in besedilnega modela smo s stikom združili v matriko:

```
#Združitev izhodov
concat_img_caption = concatenate([repeat_img, cap_dense],
name='concat_img_caption_vecs')
#Zadnji LSTM z 1000 enotami
final_lstm = LSTM(1000, return_sequences=False,
name='many2one_final_lstm')(concat_img_caption)
final_dropout = Dropout(0.1)(final_lstm)
#Napovedi razredov (besed)
final_dense = Dense(vocab_size,
name='final_dense')(final_dropout)
prediction = Activation('softmax',
name='final_prediction')(final_dense)
#Sestavimo model z vhodi in izhodi
model = Model(inputs=[image_input, caption_input],
outputs=prediction)
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

Združena izhoda smo povezali v vhod zadnjega sloja rekurenčne nevronske mreže LSTM. Obdelovali smo jih kot zaporedje vrednosti. Zaradi tega se je zadnji sloj LSTM naučil, katere besede generirati ob značilkah trenutne vhodne slike.

Zadnji sloj arhitekture je polno povezana nevronska mreža, ki iz prejetega izhoda LSTM sloja, napove verjetnosti razredov (besed). Model smo sestavili s funkcijo *Model*, v kateri smo nanizali vhodna vozlišča in zadnji sloj (izhod). S parametri *loss* in *optimizer* smo določili funkcijo, ki je izračunala napako, in optimizacijski algoritem. Za izračun napake smo uporabili cenilko križne entropije. Optimizacijo parametrov opisujemo v razdelku 5.2.

Poglavje 5

Učenje in ocenjevanje modela

V tem poglavju si bomo pogledali način priprave vhodnih podatkov in postopek preizkušanja različnih parametrov modela med učenjem. Predstavili bomo nekaj generiranih opisov in ocenili delovanje modela z mero BLEU [29].

5.1 Predpriprava podatkov

V tem razdelku se bomo osredotočili na obliko in način predprocesiranja vhodnih podatkov za učenje modela. Spoznali bomo, kako je potekalo učenje in različne kombinacije parametrov, ki smo jih prilagajali za izboljšanje rezultatov. Predstavili bomo nekaj rezultatov modela in ga ocenili.

Model konvolucijske nevronske mreže VGG16 smo ločili od končne arhitekture našega modela, da smo pospešili učenje in napovedovanje modela. Odločili smo se, da bo končni model prejemal predhodno obdelane podatke iz VGG16 modela in bo za vhodne vrednosti prejel vektor značilk dolžine 4096.

Slike iz podatkovne zbirke MS COCO smo razdelili na dve podmapi: učno in testno množico. Nato smo postopoma slike v pythonu prebirali iz direktorija in jih obdelovali. VGG16 arhitektura je zahtevala, da vhodni sliki preoblikujemo dimenzije in jo pravilno skaliramo. Dimenzije slik smo takole pripravili za VGG16 model:

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np
def preprocess_image(img_path, target_size = (224 ,224)):
    img = image.load_img( img_path,target_size = target_size )
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    img = preprocess_input(img)
    return np.asarray(img)
```

Pridobljene vektorje značilk smo shranili v podatkovno strukturo slovarja, v katerem je ključ predstavljalo ime datoteke (slike), vrednost pa vektor značilk slike, dimenzij $1 * 4096$:

```
def encode_img(preprocessed_img, model):
    features = model.predict(preprocessed_img)
    features_vec = np.reshape(features, features.shape[1])
    return features_vec
```

Slovar vektorjev značilk slik smo z uporabo knjižnice *pickle* shranili v datoteko za kasnejšo ponovno uporabo.

Sledila je priprava opisov in slovarja učnih besed. Iz datoteke dodatnih informacij slik podatkovne zbirke, zapisane v json formatu, smo prebrali vrednosti opisov vsake slike. Ker ima slika 5 opisov in bi uporaba vseh pri učenju podaljšala postopek, smo uporabili le prvega. Opise smo razdelili na posamezne unikatne besede in s tem ustvarili besedne žetone (ang. word tokens). Vsaki besedi smo prešteli število pojavitev v celotnem korpusu opisov in vrednosti shranili v podatkovno strukturo slovarja. Celotna množica unikatnih besed, pridobljenih iz zbirke opisov, je bila velika približno 28 tisoč besed. Ker je množico sestavljalo veliko število besed, jo je bilo zaradi kakovosti in hitrosti učenja potrebno reducirati. Izkazalo se je, da je pogosto uporabljenih besed le nekaj tisoč, ostale so redkeje pojavljajoče se besede korpusa ali tipkarske napake v opisih. Da smo se izognili šumu in smiselno reducirali

velikost slovarja unikatnih besed, smo najprej slovar razvrstili padajoče po frekvenci pojavitev besed. Razvrščenemu slovarju smo zmanjšali velikost na 15 tisoč besed, poskusili smo tudi z 10200 besedami. S tem postopkom smo izpustili besede, ki se malokrat pojavijo v opisih in nimajo velikega vpliva pri učenju modela.

Slovar besed smo indeksirali. Ker smo se odločili za uporabo vektorske vložitve besed z GloVe vektorji, smo besedam našega slovarju poiskali inačico besede v slovarju GloVe. Če se beseda iz našega slovarja ni pojavila v slovarju GloVe, smo ji generirali naključen vektor dolžine 200. Obenem smo ustvarili še dva dodatna slovarja in *seznam vložitvenih vektorjev*, pridobljen iz slovarja GloVe. Prvi slovar je zaznamoval preslikavo besede v indeks, kjer je ključ slovarja beseda, vrednost pa indeks, na katerem se v seznamu vložitvenih vektorjev nahaja vektor določene besede. Drugi slovar je deloval kot obratna preslikava indeksa v besedo. Slovarja sta bila uporabljena pri kodiranju vhodov podatkov opisa in dekodiranju napovedi našega modela. Seznam vložitvenih vektorjev je bil predložen vložitvenemu sloju *besedilnega modela*, opisanega v razdelku 4.2.2.

V slovar unikatnih besed smo dodali dva posebna žetona <*begin*> in <*end*>. Prvi žeton je bil dodan na začetek vsakega opisa. Njegova naloga je zagotoviti, da se model s tem žetonom nauči tvoriti začetek stavka. S to tehniko ustvarimo začetek generiranja besed v zaporedju, ki jih podrobneje opišemo v razdelku 5.3. Drugi žeton smo dodali na konec in nakazuje konec generiranje opisa. Zanj želimo, da ima visoko verjetnost pojavitve pri zaključku opisnega stavka.

Implementirali smo generator podatkov in pripravljene podatke postopoma pošiljali na vhodna vozlišča nevronskega modela. Za dano sliko smo v slovarju značilk slik poiskali pripadajoč vektor značilk in ga ovili v numpyev seznam. V drugem koraku smo poiskali nekodiran (besedni) opis dane slike. Opis smo razbili na posamezne besede in za vsako besedo:

1. Poiskali preslikavo iz besede v indeks.

2. Ustvarili numpyev seznam indeksov vseh prejšnjih besed opisa, vključno

s trenutno preslikavo besede.

3. Ustvarili ničelni numpyev seznam v velikosti našega slovarja besed.

4. Poiskali indeks naslednje besede trenutnega opisa in v ničelnem (binarnem) seznamu na istoležnem indeksu nastavili vrednost na 1.

5. Seznam indeksov trenutne sekvence besed dopolnili z ničlami do dolžine najdaljšega opisa.

6. Vektor značilk slike in seznam indeksiranih sekvenc predložili na vhod modela, na izhod pa numpyev binarni seznam ničel in enega pozitivnega indeksa (razreda) željene napovedane besede.

Za lažjo predstavo delovanja si oglejmo primer. Učni primer je slika žirafe, ki stoji ob drevesu. Pripadajoč opis: *<begin> a giraffe is standing next to a tree <end>*. Slovar preslikav besed v indeks je lahko sledeč:

```
{"a": 2, "giraffe": 4, "is": 3, ...}
```

Po opisu iteriramo besedo za besedo. V prvi iteraciji:

```
1. word2ind = 2
2. sub_sequence = [ 2 ]
3. predict = [ 0, 0, 0, ..., 0]
4. giraffe_indeks = 4
   predict[giraffe_indeks] = 1
5. sub_sequence = [ 2 , 0, 0, ..., 0]
6. model.input( [image_features, sub_sequence] )
   model.predict( predict )
```

Postopek ponavaljamo do predzadnje besede v trenutnem opisu.

5.2 Izbira parametrov arhitekture

Model smo po postopku, opisanem v prejšnjem razdelku 5.1, učili na učni množici 82.783 slik in opisov. Med učenjem smo spremljali napako modela, merjeno s funkcijo križne entropije:

$$H(p, q) = -\sum_{x} p(x) \log q(x), \tag{5.1}$$

kjer je $p(x)$ dejanska vrednost razreda x in $q(x)$ napovedana verjetnost razreda x [31]. Napake so predstavljene na Sliki 5.2 in 5.3. Klasifikacijsko točnost smo merili z izrazom:

$$ACC = \frac{TP + TN}{P + N}, \tag{5.2}$$

kjer sta TP (ang. true positives) in TN (ang. true negatives) pravilne razvrstitve vzorcev, P (ang. positives) in N (ang. negatives) pa število vseh vzorcev [30].

Napovedovali smo razrede, kjer vsakega predstavlja ena beseda. Ker je večkrat prišlo do visokih vrednosti napak, smo preiskali nabor parametrov modela, ki so navedeni na Sliki 5.1. Vsako konfiguracijo parametrov smo zaznamovali s številko modela in ga učili nekaj epoh. Če se je izkazalo, da je konfiguracija privedla do stagniranja, premajhnega ali prevelikega prileganja učni množici, smo ga predčasno ustavili in preizkusili novo konfiguracijo. Model smo učili z uporabo grafične kartice *Nvidia GTX 970* z 1664 jedri CUDA, 1114 megaherčnim taktom procesorja, 1750 megaherčnim taktom pomnilnika in 4096 megabajtnim grafičnim pomnilnikom. V povprečju je ena učna epoha trajala 2 uri.

Stolpič *število enot LSTM* na Sliki 5.1 se navezuje na število enot, uporabljenih v zadnjem LSTM sloju modela (Slika 4.2) po združitvi izhodov. *Število slojev LSTM* in *število slojev izpustitev nevronov* se navezujejo na število vseh takih slojev v arhitekturi.

Optimizator se je izkazal kot najvplivnejši parameter, saj je s svojim učnim faktorjem vplival na hitrost učenja modela. Optimizator Adam se je učil hitreje kot RMSprop, vendar je pri njegovi uporabi večkrat prišlo do prevelikega

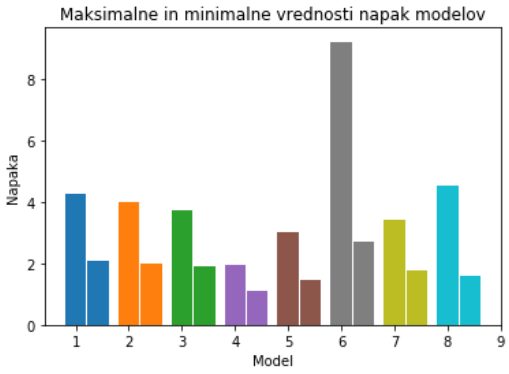
Št. modela	Optimizator	Učni faktor	Vložitev GloVe	Dolžina vektorja vložitve	Število enot LSTM	Število slojev LSTM	Število slojev izpustitev nevronov	Velikost slovarja (število razredov)	Velikost učnega paketa
1	RMSprop	0.001	DA	200	1000	1	2	15000	360
2	RMSprop	0.001	DA	200	1000	2	4	15000	360
3	RMSprop	0.001	DA	200	1000	4	4	15000	360
4	Adam	0.01	DA	200	1000	2	3	10200	400
5	Adam	0.01	DA	200	1000	3	4	10200	400
6	RMSprop	0.001	DA	200	780	3	5	10200	400
7	Adam	0.01	DA	200	1500	2	4	10200	256
8	Adam	0.01	NE	100	1500	2	4	10200	128

Slika 5.1: Različni parametri uporabljeni pri učenju modelov.

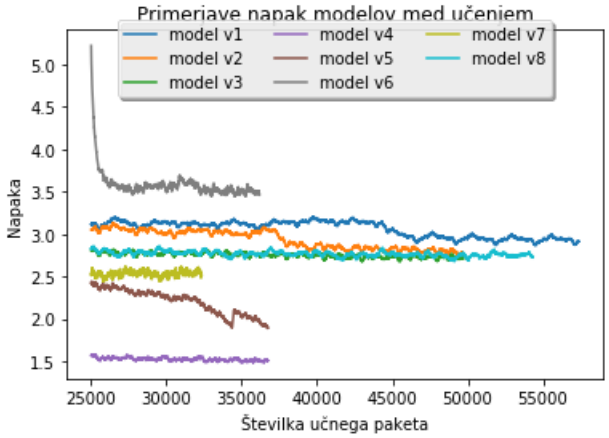
prileganja učni množici. RMSprop je posodabljal uteži modela počasneje, posledično se je počasi približeval globalnemu minimumu napake. Kadar smo uporabili večje število slojev izpustitev nevronov, je model že na začetku in tudi v kasnejši fazi učenja dosegal premajhno prileganje učni množici. *Število slojev LSTM* in *velikost učnega paketa* nista dosti vplivala na natančnost napovedi modela, pač pa na hitrost iteracije epohe. Večje kot je bilo število LSTM slojev, več prilagodljivih parametrov je imela arhitektura modela, daljša je bila izračunava vseh uteži in iteriranje epohe. Večji, kot je bil učni paket, hitreje smo opravili iteracijo celotne učne množice, vendar nas je pri tem parametru omejevala velikost pomnilnika grafične kartice. Večje število enot v zadnjem LSTM sloju je zmanjšalo napako modela, vendar se je z večimi parametri ponovno povečala kompleksnost arhitekture.

Pri modelu številka 8 nismo uporabili naučenih vložitvenih vektorjev, zato se jih je moral naučiti sam. Klasifikacijska točnost modela je bila posledično že na začetku učenja veliko slabša od ostalih. Zmanjšanje velikosti *slovarja* je zmanjšalo napako modela (5.1) za vrednost ≈ 0.8, saj smo s tem zmanjšali število razredov (besed) napovedi, brez velikih izgub pri informaciji o opisih. Najboljša konfiguracija parametrov se je izkazala za model številka 4, ki smo ga uporabili kot končnega in ga podrobneje ovrednotili in preizkusili. Učili smo ga 65 epoh (približno 5 dni neprekinjeno). Dosegel je manjšo napako od ostalih (minimalna vrednost 1.3) in klasifikacijsko točnost ≈ 60 % (5.2).

Najslabše se je odrezal model številka 6, ki je zaradi preveč slojev izpustitev nevronov dosegel premajhno prileganje učni množici in klasifikacijsko točnostjo ≈ 43 % in se prepočasi približeval minimalni vrednosti napake modela. Klasifikacijska točnost in napaka v našem primeru nista dajala očitnega odgovora o kvaliteti opisov, zato smo najboljši model ocenili z mero BLEU.



Slika 5.2: Primerjava maksimalnih in minimalnih vrednosti napak modelov z različnimi parametri na koncu učenja.



Slika 5.3: Primerjava napak modelov z različnimi parametri med učenjem.

Vrednosti napake (os y) na Sliki 5.2 in 5.3 so bile izračunane s funkcijo križne entropije (5.1).

5.3 Generiranje opisov

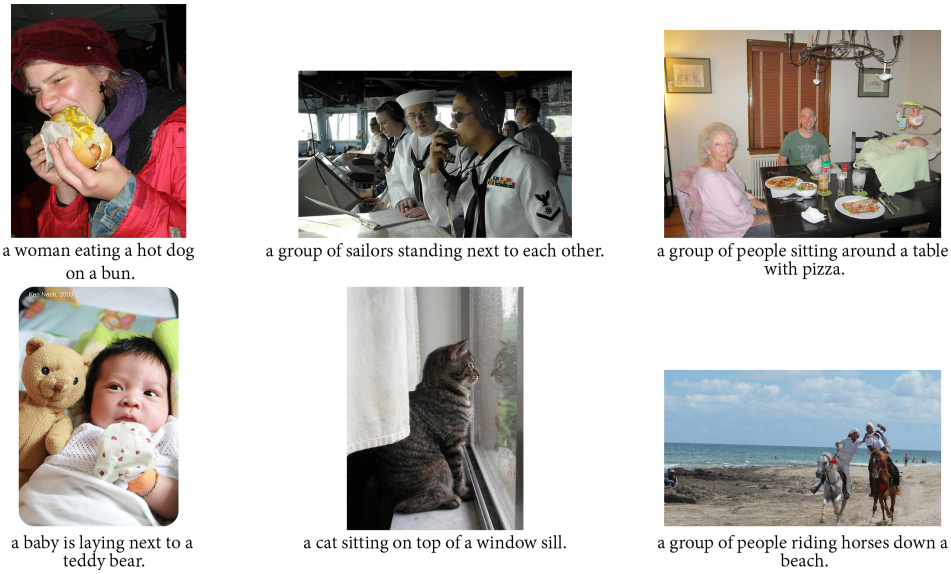
Postopek generiranja opisa slik smo začeli z vektorjem značilk slike, ki smo ga pridobili iz slovarja značilk slik ali pa ga generirali ponovno za novo sliko, ki smo jo najprej skalirali na prave dimenzije in poslali na vhod modificiranega VGG16 modela. Vektor značilk slike smo na način, opisan v prejšnem razdelku, ovili v numpyev seznam. Nato smo ustvarili nov seznam, v katerega smo shranjevali podzaporedja generiranih besed.

V seznam podzaporedij smo vstavili začetno podzaporedje, v katerem je bila zapisana le ena beseda oziroma indeks žetona <*begin*>. Modelu smo na vhodna vozlišča v zanki dostavljali vektor značilk slike in seznam podzaporedij, ob vsaki napovedi smo iz napovedanih indeksov besed (razredov) izbrali najbolj verjetnega in ga pripeli v vhodno podzaporedje ter postopek ponavljali. Na ta način smo sestavljali zaporedje indeksov besed, dokler nismo sestavili zaporedja želene dolžine. Končno zaporedje indeksov besed smo nato s pomočjo slovarja preslikave indeksa, v besedo združili v besedno zaporedje (stavek). Generiranem opisu smo za konec odstranili začetni in končni žeton (<*begin*> in <*end*>). Postopek smo ponovili za vse slike testne množice. Iz rezultatov smo izbrali nekaj zanimivejših slik (5.4, 5.5 in 5.6).

Kljub dokaj slabi klasifikacijski točnosti in veliki napaki, je model znal tvoriti pravilne angleške stavke. Pozna semantiko besed in stavčno strukturo, zna pravilno uvrstiti predloge, glagole in samostalnike. Večkrat se je izkazalo, da deloma razume vsebino slike in poizkuša tvoriti stavek iz besed, ki se jih je naučil na podobnih vsebinah slik in imajo nekaj skupnega z dano sliko. Kadar ima slika veliko podrobnosti ali različnih objektov, model največkrat nepravilno prepozna vsebino, saj konvolucijske nevronske mreže razbirajo vsebino slik na teksturni ravni.

5.4 Ocenitev opisov z mero BLEU

Za ocenjevanje rezultatov modela smo uporabili mero BLEU (ang. bilingual evaluation understudy), s pomočjo katere smo dobili boljši vpogled v



Slika 5.4: Primeri dobrih opisov slik.

kvaliteto generiranih opisov. Namen mere BLEU je ocenjevanje kvalitete računalniških jezikovnih prevodov [29]. Mera BLEU je algoritem, ki v danem besedilu šteje, koliko besed se ujema z referenčnim besedilom (idealnim prevodom). V praksi se uporablja tako, da računalniško prevedena besedila primerjamo z referenčnimi človeškimi prevodi besedila. Algoritem uporabi dano besedilo (kandidat) in jo primerja z referenčnim z izrazom:

$$P = \sum_{i=1}^{n} \frac{\min(m_i, m_{i(refmax)})}{w_t}, \tag{5.3}$$

kjer je n število unikatnih besed v kandidatu (besedilu), m_i število pojavitev i-te unikatne besede kandidata v referenčnih besedilih, $m_{i(refmax)}$ največje število pojavitev i-te unikatne besede kandidata v referenčnih besedilih in w_t število vseh besed kandidata.

Ker naš model generira besedne opise slik, smo jih uporabili kot kandidate, prave opise pa kot referenčne. Generirane opise smo ocenili z mero BLEU na stavčnem nivoju in dosegli 49.0, kar velja za precej dobro vrednost (več je bolje).

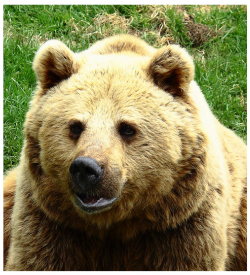
Slika 5.5: Primeri slabših opisov slik.

Rezultat smo primerjali z modelom Andreja Karpathya, ki je dosegel 62.5 [12], Googlovim modelom Show and Tell z 66.6 [28] in modelom LRCN z vrednostjo 62.8 [5]. Našteti modeli so dosegli dokaj boljšo mero BLEU, kjer Googlov dosega najboljši rezultat in se z njim najbolj približajo povprečni vrednosti mere BLEU človeka – 69.0 [28].

a little girl holding a green tie in their right hand.

an elephant standing next to a man on a sidewalk.

a potted tree in the middle of a garden.



a couple of brown bears standing next to each other.

a pair of scissors sitting next to a pair of scissors.

a man standing on a city street holding a surfboard.

Slika 5.6: Primeri zelo slabih opisov slik.

Poglavje 6

Sklepne ugotovitve

V diplomskem delu smo pregledali teorijo globokega učenja in različne tipe nevronskih mrež, ki smo jih uporabili pri izdelavi arhitekture modela za generiranje opisov iz slik. Uporabili smo podobne tehnike, ki so bile uporabljene in opisane v delu Karpathya in Googlovem projektu *Show and Tell*. Pridobili smo slike in opise iz MS COCO podatkovne zbirke. Podatke smo predhodno obdelali, besedila preslikali v vektorski prostor s pomočjo uporabe GloVe vložitvenih vektorjev, iz slik smo izluščili značilke s pomočjo konvolucijske nevronske mreže VGG16. Arhitekturo (Slika 4.2) smo implementirali v pythonovi knjižnici Keras. V arhitekturi smo uporabili rekurenčne nevronske mreže LSTM za učenje stavkov. Modelu smo pripravili vhodne podatke in ga učili. Spremljali smo njegovo napako ter klasifikacijsko točnost in spreminjali različne parametre, delno tudi arhitekturo, da smo dosegli čim boljše rezultate učenja. Izbrali smo si parametre, ki so na modelu dosegli najboljše rezultate in model še naprej učili, testirali in ocenili. Po koncu učenja smo dosegli klasifikacijsko točnost ≈ 60 %, napako ≈ 1.3 in mero BLEU na stavčnem nivoju 49.0, kar velja za solidno oceno. Rezultat mere BLEU ni primerljiv z rezultatom Googlovega Show and Tell modela z vrednostjo 66.6. Razvita tehnologija se lahko uporablja za izboljšanje rezultatov spletnih brskalnikov, pri označevanju in opisovanju vsebine slik, na področju računalniškega vida pri detekciji predmetov, pomoč slepim in slabovidnim pri razumevanju

slikovnih vsebin ipd.

Model ima še veliko prostora za izboljšave. Z modelom ne moremo opisati katerekoli slike, ker ga je potrebno precej časa učiti na želeni problemski domeni z velikim številom učnih primerov. Model uporablja konvolucijsko nevronsko mrežo VGG16, ki izlušči samo značilke slik in ne klasificira ali detektira objektov na sliki. Možno ga je izboljšati na več načinov. Pri tvorjenju opisov ne uporabljamo podatka o pozornosti (ang. attention) in ne uporabljamo vseh 5 opisov slike. Model bi lahko učili več časa, z boljšo strojno opremo in preizkušali druge kombinacije parametrov ali celotno arhitekturo naredili kompleksnejšo/globljo. Lahko bi uporabili tudi druge vložitve besed in večjo dolžino vložitvenih vektorjev.

Literatura

[1] Continuum. What is Anaconda? Dosegljivo: `https://www.continuum.io/what-is-anaconda`, 2017. [Dostopano 09. 08. 2017].

[2] NumPy developers. NumPy. Dosegljivo: `http://www.numpy.org/`, 2017. [Dostopano 09. 08. 2017].

[3] Pandas developers. Pandas. Dosegljivo: `http://pandas.pydata.org/`, 2017. [Dostopano 09. 08. 2017].

[4] Pickle developers. Pickle documentation. Dosegljivo: `https://docs.python.org/2/library/pickle.html`, 2017. [Dostopano 09. 08. 2017].

[5] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[6] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *European conference on computer vision*, pages 15–29. Springer, 2010.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[10] ImageNet. About ImageNet. Dosegljivo: `http://image-net.org/about-overview`, 2017. [Dostopano 09. 08. 2017].

[11] Christopher D. Manning Jeffrey Pennington, Richard Socher. GloVe. Dosegljivo: `https://nlp.stanford.edu/projects/glove/`, 2017. [Dostopano 09. 08. 2017].

[12] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[13] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[14] Andrey Karpathy. Website of Deep Visual-Semantic Alignments for Generating Image Descriptions. Dosegljivo: `http://cs.stanford.edu/people/karpathy/deepimagesent/`, 2015. [Dostopano 22. 07. 2017].

[15] Keras. Keras Documentation. Dosegljivo: `https://keras.io/`, 2017. [Dostopano 09. 08. 2017].

[16] Girish Kulkarni, Visruth Premraj, Vicente Ordonez, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, and Tamara L Berg. Babytalk: Understanding and generating simple image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2891–2903, 2013.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[19] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft CoCo: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[20] Microsoft. MS COCO. Dosegljivo: `http://mscoco.org/dataset/#download`, 2017. [Dostopano 09. 08. 2017].

[21] Michael Nielsen. How the backpropagation algorithm works. Dosegljivo: `http://neuralnetworksanddeeplearning.com/chap2.html`, 2017. [Dostopano 09. 08. 2017].

[22] Nvidia. CUDA Toolkit. Dosegljivo: `https://developer.nvidia.com/cuda-toolkit`, 2017. [Dostopano 09. 08. 2017].

[23] Christopher Olah. Understanding LSTM networks. *GITHUB blog, posted on August*, 27:2015, 2015.

[24] University Of Oxford. Visual Geometry Group. Dosegljivo: `http://www.robots.ox.ac.uk/~vgg/research/very_deep/`, 2017. [Dostopano 09. 08. 2017].

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Research Repository*, abs/1409.1556, 2014.

[26] Sai Tai. Image Classification. Dosegljivo: http://www.sai-tai.com/blog/2017/04/12/image-classification/, 2017. [Dostopano 09. 08. 2017].

[27] PennState University. Neurons. Dosegljivo: https://online.science.psu.edu/bisc004_activewd001/node/1907, 2017. [Dostopano 22. 07. 2017].

[28] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[29] Wikipedia. BLEU. Dosegljivo: https://en.wikipedia.org/wiki/BLEU, 2017. [Dostopano 09. 08. 2017].

[30] Wikipedia. Confusion matrix. Dosegljivo: https://en.wikipedia.org/wiki/Confusion_matrix, 2017. [Dostopano 09. 08. 2017].

[31] Wikipedia. Cross entropy. Dosegljivo: https://en.wikipedia.org/wiki/Cross_entropy, 2017. [Dostopano 09. 08. 2017].

[32] Wikipedia. Deep learning. Dosegljivo: https://en.wikipedia.org/wiki/Deep_learning#History, 2017. [Dostopano 22. 07. 2017].

[33] Jianxin Wu. Convolutional neural networks. In *Introduction to Convolutional Neural Networks*, pages 5–23. LAMBDA Group, 2017.